Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Parallel Computing 29 (2003) 591–606

PARALLEL COMPUTING

www.elsevier.com/locate/parco

A parallel hybrid heuristic for the multicommodity capacitated location problem with balancing requirements

Bernard Gendron [a,b], Jean-Yves Potvin [a,b,*], Patrick Soriano [a,c]

[a] *Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Que., Canada H3C 3J7*
[b] *Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Que., Canada H3C 3J7*
[c] *Ecole des Hautes Etudes Commerciales 3000, chemin de la Côte-Sainte-Catherine, Montréal, Que., Canada H3T 2A7*

Received 30 May 2002; accepted 5 December 2002

Abstract

In this paper, a parallel hybrid heuristic is developed for the multicommodity capacitated location problem with balancing requirements. The hybrid involves variable neighborhood descent (VND) and slope scaling (SS). Both methods evolve in parallel within a master–slave architecture where the slave processes communicate through adaptive memories. Numerical results are reported on different types of randomly generated instances, using an increasing number of processors and different distributions of processes between SS and VND.
© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Multicommodity capacitated location problem; Slope scaling; Variable neighborhood descent; Hybrid; Adaptive memory

1. Introduction

This paper describes a parallel approach for solving a variant of the multicommodity location problem with balancing requirements [2]. This problem comes from

* Corresponding author. Address: Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Que., Canada H3C 3J7.
E-mail address: potvin@iro.umontreal.ca (J.-Y. Potvin).

0167-8191/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0167-8191(03)00044-9

an industrial application related to the management of a heterogeneous fleet of containers for an international maritime shipping company. Once a ship arrives at the port, the company has to deliver the loaded containers, which may come in several types and sizes, to designated in-land destinations. Following their unloading by the importing customer, empty containers are moved to a depot. Later on, they may be delivered to customers requesting containers for subsequent shipping of their own products. Due to regional disparities in empty container availabilities and needs throughout the network, balancing movements of empty containers among the depots are required. This characteristic differentiates the problem from classical location–allocation applications.

Thus, the problem is to locate the depots which collect the supply of empty containers to satisfy the demand for empty containers, while minimizing the total operating costs. These include the cost of opening and operating the depots and the cost generated by customer–depot and interdepot movements. Different exact and heuristic approaches have been proposed in the past for the uncapacitated version of this problem [3–7,9]. However, a more challenging capacitated version, where each depot has a fixed and finite capacity, has only been recently addressed in [8] (a review of other classes of capacitated facility location problems can be found in [16]). The present paper proposes a parallel search framework for the same problem where variable neighborhood descent (VND) and slope scaling (SS) processes communicate through adaptive memories. Although presented in the context of a particular application, the proposed framework could be used to solve other complex optimization problems, through the involvement of different types of cooperating heuristics.

The organization of the paper is the following. In Section 2, a mathematical formulation of the multicommodity capacitated location problem with balancing requirements (MCLB) is proposed. This formulation is used to isolate multicommodity minimum cost network flow subproblems (MMCFs), which are exploited by SS and VND. These two heuristics are then briefly sketched in Section 3. A parallel implementation of a hybrid framework involving both heuristics is then presented in Section 4. Computational experiments on randomly generated test problems with different characteristics are reported in Section 5, using an increasing number of processors and different distributions of processes between SS and VND. Concluding remarks follow in Section 6.

2. Problem formulation

To formulate the problem, we consider a directed network $G = (N, A)$, where N is the set of nodes and A is the set of arcs. There are several commodities (types of containers) moving through the network which are represented by set K. The set of nodes can be partitioned into the set of customer nodes C and the set of depots D. For each depot $j \in D$, we define its supply and demand customers as $C_j^s = \{i \in C : (i,j) \in A\}$ and $C_j^d = \{i \in C : (j,i) \in A\}$, respectively. We also assume that there is at least one supply or demand customer adjacent to each depot, that is, $C_j^s \cup C_j^d \neq \emptyset, \forall j \in D$. The sets of all supply and demand customers thus correspond

to $C^s = \bigcup_{j\in D} C_j^s$ and $C^d = \bigcup_{j\in D} C_j^d$, respectively. For each node $i \in N$ (depot or customer), we also define the sets of depots adjacent to this node in both directions $D_i^- = \{j \in D : (j,i) \in A\}$ and $D_i^+ = \{j \in D : (i,j) \in A\}$.

Since there are no arcs between pairs of customers, the set of arcs can be partitioned into three subsets:

- customer-to-depot arcs $A_{\mathrm{CD}} = \{(i,j) \in A : i \in C,\ j \in D\}$;
- depot-to-customer arcs $A_{\mathrm{DC}} = \{(j,i) \in A : j \in D,\ i \in C\}$;
- depot-to-depot arcs $A_{\mathrm{DD}} = \{(l,j) \in A : l \in D,\ j \in D\}$.

The problem consists of minimizing the costs incurred by moving flows of commodities through the network to satisfy the supplies at origins and the demands at destinations. For each customer $i \in C^s$, the supply of commodity k is noted s_i^k, while for each customer $i \in C^d$, the demand for commodity k is noted d_i^k. All supplies and demands are assumed to be non-negative and deterministic. A non-negative cost c_{ij}^k is incurred for each unit of flow of commodity k moving on arc (i,j). In addition, for each depot $j \in D$, a non-negative fixed cost f_j is incurred if the depot is opened. The problem is further complicated by the presence of a fixed capacity q_j on the volume of all commodities which can transit through depot $j \in D$, where the volume of one unit of commodity k is noted v_k.

Let x_{ij}^k represent the flow of commodity k moving on arc (i,j), and y_j be the binary location variable with value 1 if depot j is opened, and value 0 otherwise. The problem is then formulated as:

$$Z = \min \sum_{j\in D} f_j y_j + \sum_{k\in K} \left(\sum_{(i,j)\in A_{\mathrm{CD}}} c_{ij}^k x_{ij}^k + \sum_{(j,i)\in A_{\mathrm{DC}}} c_{ji}^k x_{ji}^k + \sum_{(l,j)\in A_{\mathrm{DD}}} c_{lj}^k x_{lj}^k \right), \tag{1}$$

subject to

$$\sum_{j\in D_i^+} x_{ij}^k = s_i^k, \quad \forall i \in C^s,\ k \in K, \tag{2}$$

$$\sum_{j\in D_i^-} x_{ji}^k = d_i^k, \quad \forall i \in C^d,\ k \in K, \tag{3}$$

$$\sum_{i\in C_j^d} x_{ji}^k + \sum_{l\in D_j^+} x_{jl}^k - \sum_{i\in C_j^s} x_{ij}^k - \sum_{l\in D_j^-} x_{lj}^k = 0, \quad \forall j \in D,\ k \in K, \tag{4}$$

$$\sum_{k\in K} v^k \left(\sum_{i\in C_j^s} x_{ij}^k + \sum_{l\in D_j^-} x_{lj}^k \right) \leqslant q_j y_j, \quad \forall j \in D, \tag{5}$$

$$x_{ij}^k \geqslant 0, \quad \forall (i,j) \in A,\ k \in K, \tag{6}$$

$$y_j \in \{0,1\}, \quad \forall j \in D. \tag{7}$$

Constraints (2) and (3) ensure that supply and demand requirements are met. Constraints (4) and (5) are the flow conservation constraints and capacity constraints for each depot, respectively. Constraints (5) also forbid customer-related movements through closed depots.

The following constraints, (8) and (9), which are redundant, are also added to the above formulation when solving large-scale instances of the MCLB with a state-of-the-art MIP code (see Section 5). They have proven to significantly improve the quality of the lower bounds obtained through the LP relaxation.

$$x_{ij}^k \leqslant s_i^k y_j, \quad \forall j \in D,\ \ i \in C_j^s,\ \ k \in K, \tag{8}$$

$$x_{ji}^k \leqslant d_i^k y_j, \quad \forall j \in D,\ \ i \in C_j^d,\ \ k \in K. \tag{9}$$

Upper bounds on the optimal value of the MCLB can be derived by fixing both the vector of location variables y to some value $\bar{y}$ and the vector of linear costs c to some value $\bar{c}$. We then obtain the following MMCF:

$$\min \sum_{k\in K}\left(\sum_{(i,j)\in A_{\mathrm{CD}}} \bar{c}_{ij}^k x_{ij}^k + \sum_{(j,i)\in A_{\mathrm{DC}}} \bar{c}_{ji}^k x_{ji}^k + \sum_{(l,j)\in A_{\mathrm{DD}}} \bar{c}_{lj}^k x_{lj}^k\right), \tag{10}$$

subject to constraints (2)–(4), (6) and

$$\sum_{k\in K} v^k\left(\sum_{i\in C_j^s} x_{ij}^k + \sum_{l\in D_j^-} x_{lj}^k\right) \leqslant q_j \bar{y}_j, \quad \forall j \in D. \tag{11}$$

Assuming that this problem has an optimal solution, $\tilde{x}$, an upper bound on the optimal value of the MCLB is then given by:

$$Z(\tilde{x},\tilde{y}) = \sum_{j\in D} f_j \tilde{y}_j + \sum_{k\in K}\left(\sum_{(i,j)\in A_{\mathrm{CD}}} c_{ij}^k \tilde{x}_{ij}^k + \sum_{(j,i)\in A_{\mathrm{DC}}} c_{ji}^k \tilde{x}_{ji}^k + \sum_{(l,j)\in A_{\mathrm{DD}}} c_{lj}^k \tilde{x}_{jl}^k\right), \tag{12}$$

where

$$\tilde{y}_j = \begin{cases} 1, & \text{if } \left(\sum_{i\in C_j^s} \tilde{x}_{ij}^k + \sum_{l\in D_j^-} \tilde{x}_{lj}^k\right) > 0, \quad \forall j \in D. \\ 0, & \text{otherwise,} \end{cases} \tag{13}$$

The SS procedure considers MMCFs with all depots open (i.e., $\bar{y}_j = 1, \forall j \in D$), but modifies the linear costs $\bar{c}$ at each iteration. The VND method solves a series of MMCFs with $\bar{c} = c$ (i.e., the linear costs are not modified), but with y fixed to values $\bar{y}$ determined by the search procedure. Both heuristic procedures are described next.

3. Heuristics

In the sequential algorithm proposed in [8], SS is used to generate an initial starting solution for a tabu search heuristic. In the parallel framework presented here, SS

is still used, but tabu search is replaced by a less CPU-intensive VND search. In the following, we briefly describe SS, before proceeding with VND.

3.1. Slope scaling

SS is an iterative procedure, where successive MMCFs are solved, based on modified linear costs (see [12–14] for recent successful applications of this type of heuristic for solving non-convex piecewise linear network flow problems). In our context, all depots are implicitly open in the formulation of the MMCF (i.e., $\bar{y}_j = 1, \forall j \in D$), but the linear costs, $\bar{c}$, are modified to reflect the contribution of the fixed costs.

More specifically, given a solution $\tilde{x}$ to some MMCF formulation, and assuming that $\tilde{y}$ is computed according to (13), the linear costs are modified so that

$$\sum_{k\in K}\left(\sum_{(i,j)\in A_{\mathrm{CD}}}\bar{c}_{ij}^{k}\tilde{x}_{ij}^{k}+\sum_{(j,i)\in A_{\mathrm{DC}}}\bar{c}_{ji}^{k}\tilde{x}_{ji}^{k}+\sum_{(l,j)\in A_{\mathrm{DD}}}\bar{c}_{lj}^{k}\tilde{x}_{lj}^{k}\right)=Z(\tilde{x},\tilde{y}), \tag{14}$$

where $Z(\tilde{x},\tilde{y})$ is the total cost of the feasible solution $(\tilde{x},\tilde{y})$ given by Eq. (12). In other words, the goal is to solve an MMCF with modified costs $\bar{c}$ that reflect exactly the total cost, both linear and fixed, incurred by this solution (if the solution remains the same). To compute the modified costs, we use the total volume in-transit through each depot j, defined for any feasible flow $\tilde{x}$, as:

$$\widetilde{X}_j=\sum_{k\in K}v^k\left(\sum_{i\in C_j^s}\tilde{x}_{ij}^{k}+\sum_{l\in D_j^-}\tilde{x}_{lj}^{k}\right). \tag{15}$$

The modification to the linear costs proceeds as follows. We denote the modified cost associated with arc (i,j) and commodity k at iteration $t \geqslant 0$ as $\bar{c}_{ij}^{k(t)}$. Similarly, we denote by $\widetilde{X}_j^t$ the total volume in-transit through depot j at iteration $t \geqslant 0$. Initially, at iteration 0, we set the modified costs as follows:

$$\bar{c}_{ij}^{k(0)}=c_{ij}^{k}+\lambda v^k\alpha_j\frac{f_j}{q_j},\quad \forall(i,j)\in A_{\mathrm{CD}},\ \ k\in K, \tag{16}$$

$$\bar{c}_{ji}^{k(0)}=c_{ji}^{k}+\lambda v^k(1-\alpha_j)\frac{f_j}{q_j},\quad \forall(j,i)\in A_{\mathrm{DC}},\ \ k\in K, \tag{17}$$

$$\bar{c}_{lj}^{k(0)}=c_{lj}^{k}+\lambda v^k\left(\alpha_j\frac{f_j}{q_j}+(1-\alpha_l)\frac{f_l}{q_l}\right),\quad \forall(l,j)\in A_{\mathrm{DD}},\ \ k\in K, \tag{18}$$

where λ is a parameter and

$$\alpha_j=\frac{\xi_j}{\xi_j+\Delta_j};\quad \xi_j=\sum_{k\in K}v^k\sum_{i\in C_j^s}s_i^k;\quad \Delta_j=\sum_{k\in K}v^k\sum_{i\in C_j^d}d_i^k,\quad \forall j\in D.$$

Here, $\xi_j(\Delta_j)$ is the maximum volume that might transit through depot j from all supply (demand) customers adjacent to it. Thus, α_j and $(1-\alpha_j)$ approximate the fraction of the total volume in-transit at depot j which can be imputed to supply and

demand customers, respectively. They are used to calibrate the cost on the corresponding arcs. In the sequential algorithm reported in [8], λ is set to 1, while in the parallel algorithm described in Section 4, parameter λ takes different values in the interval [0,1] in order to generate different starting solutions.

Based on the optimal solution to the MMCF with modified costs obtained at the previous iteration, the linear costs are updated as follows at every iteration $t > 0$. First, we consider the arcs which are incident only to depots that are used for in-transit flows (i.e., $X_j^{t-1} > 0$ for any depot j at the initial or terminal end of the arc):

$$\bar{c}_{ij}^{k(t)} = c_{ij}^k + v^k \alpha_j \frac{f_j}{\widetilde{X}_j^{t-1}}, \quad \forall (i,j) \in A_{\mathrm{CD}}, \ k \in K, \tag{19}$$

$$\bar{c}_{ji}^{k(t)} = c_{ji}^k + v^k (1-\alpha_j) \frac{f_j}{\widetilde{X}_j^{t-1}}, \quad \forall (j,i) \in A_{\mathrm{DC}}, \ k \in K, \tag{20}$$

$$\bar{c}_{lj}^{k(t)} = c_{lj}^k + v^k \left(\alpha_j \frac{f_j}{\widetilde{X}_j^{t-1}} + (1-\alpha_l) \frac{f_l}{\widetilde{X}_l^{t-1}} \right), \quad \forall (l,j) \in A_{\mathrm{DD}}, \ k \in K. \tag{21}$$

It is easy to verify that this cost update satisfies (14). For arcs with *at least one* unused incident depot, we cannot apply formulas (19)–(21), since $\widetilde{X}_j^{t-1} = 0$ for at least one incident depot j. In this case, the costs of the corresponding arcs are updated as follows:

$$\bar{c}_{ij}^{k(t)} = \beta \max \left\{ \tilde{c}^{(0)}, \max_{0<\tau<t} \left\{ \bar{c}_{ij}^{k(\tau)} | \widetilde{X}_j^{\tau} > 0 \right\} \right\}, \quad \forall (i,j) \in A_{\mathrm{CD}}, \ k \in K, \tag{22}$$

$$\bar{c}_{ji}^{k(t)} = \beta \max \left\{ \tilde{c}^{(0)}, \max_{0<\tau<t} \left\{ \bar{c}_{ji}^{k(\tau)} | \widetilde{X}_j^{\tau} > 0 \right\} \right\}, \quad \forall (j,i) \in A_{\mathrm{DC}}, \ k \in K, \tag{23}$$

$$\bar{c}_{lj}^{k(t)} = \beta \max \left\{ \tilde{c}^{(0)}, \max_{0<\tau<t} \left\{ \bar{c}_{lj}^{k(\tau)} | \widetilde{X}_j^{\tau}, \widetilde{X}_l^{\tau} > 0 \right\} \right\}, \quad \forall (l,j) \in A_{\mathrm{DD}}, \ k \in K, \tag{24}$$

where $\tilde{c}^{(0)} = \max_{(i,j)\in A,\, k\in K} \bar{c}_{ij}^{k(0)}$, and $\beta > 0$ is a parameter. When $\beta = 1$, this formula sets the arc cost either to the largest cost at iteration 0 or to the largest arc cost that led to the use of all incident depots for in-transit flows in the previous iterations (a similar rule is used in [12]). Setting β to a large value virtually closes the corresponding arc; however, when β is relatively small, this decision might be reverted in the following iterations. In our implementation, β was set to 100.

The SS procedure is stopped when there are no modifications in the costs from one iteration to the next. At each iteration, a feasible solution $\tilde{x}$ is obtained, and an upper bound is computed according to Eq. (12). The best upper bound found during the search is kept in variable Z^*. A local improvement step is performed at the end of the procedure, using $\tilde{y}$, which is the depot configuration corresponding to the best overall solution. It consists in solving the MMCF with the original linear costs $\bar{c} = c$ and $\bar{y} = \tilde{y}$, thus obtaining a new feasible solution $\tilde{x}$. If $Z(\tilde{x}, \tilde{y})$ improves upon Z^*, which is often the case, it replaces it. The final solution produced by the SS procedure corresponds to the best upper bound, Z^*.

3.2. Variable neighborhood descent

In this section, familiarity with local search methods is assumed. For an introduction to local search, variable neighborhood search and VND, the reader is referred to [10]. Basically, VND explores the space of configurations of open/closed depots, by setting the y variables to 0 or 1 (see [5,8,9], for other approaches of this type in the context of tabu search). For each configuration, the flows on the customer–depot, depot–customer and depot–depot arcs are obtained by solving the associated MMCF. A particular configuration of open/closed depots with the corresponding flows represents a solution to the problem.

Since the customers are only linked to a subset of potential depot sites, some customers may not be connected to any open depot for a given configuration of depots, leading to infeasibility. Likewise, the total capacity of the open depots may be insufficient to carry the total flow. An *artificial depot* of infinite capacity is thus added to the network and linked to all customers with high arc costs, to ensure that no flow will go through these arcs, unless there is no feasible alternative. In this way, the search is allowed to explore the infeasible domain (although it always keeps track of the best feasible solution).

3.2.1. Neighborhood

Two different neighborhood structures are considered. The first neighborhood consists in moves where the value of a single y variable is set to 0 by closing a depot (DROP). Its complexity is $O(m)$, where m is the number of depots. The second neighborhood is based on SWAP moves where two variables are modified by simultaneously closing a depot and opening another one. This neighborhood is of complexity $O(m^2)$.

The motivation for this double neighborhood scheme stems from the following observations. For most problems, good solutions have approximately the same number of open depots. The DROP neighborhood is thus a good medium to find the "right" number of open depots and get close to high quality solutions. Then, the SWAP neighborhood explores alternative configurations with the number of open depots found with DROP.

The evaluation of a move, which generates a new configuration of depots, is performed by solving the associated MMCF. As this evaluation is computationally expensive, filtering techniques are developed in the case of the SWAP neighborhood. These filtering techniques are based on approximation measures which are used to rank the moves from best to worst. Then, only the top moves according to the approximation are evaluated exactly with CPLEX, among which the best one is finally taken (see [8] for a detailed description of these approximation measures). In the current implementation, 64 moves are considered, as explained in Section 5.

In the case of the DROP neighborhood, which is of lower complexity, all moves are evaluated exactly. This is desirable as every opportunity to close a depot must be exploited to generate a good solution, due to the large savings in fixed costs. This is similar to some vehicle routing applications where saving a vehicle is of premium importance.

3.2.2. Search strategy

The VND search strategy is relatively simple and corresponds to a pure local descent, but with alternating neighborhood structures. Starting from an initial solution obtained through the adaptive memories (see Section 4), it proceeds as follows:

1. Perform a descent with DROP until a local minimum is reached.
2. Perform a descent with SWAP until a local minimum is reached.
3. Repeat Steps 2 and 3 until no more improvement is obtained.

When a local minimum is reached with DROP, we thus switch to the SWAP neighborhood. This is repeated until no more improvement to the current solution is achieved.

4. A parallel hybrid framework

In this section, we propose a framework for a coarse grained master–slave parallel implementation of a hybrid algorithm using the SS and VND heuristics. This implementation is based on adaptive memories [15,17] which are aimed at providing new starting points for both SS and VND. In the case of SS, the flow structure of the starting solution is used to set the initial modified costs. In the case of VND, the starting point is used in a more standard way by providing an initial configuration of depots (i.e., a particular setting for the y variables) in the search for better configurations.

A natural way to parallelize adaptive memory algorithms is a master–slave scheme in which the master process manages the memories and a fixed number of slave processes perform the computations [1]. Basically, the master receives new solutions from the slaves, and stores them into the memories (if they are good enough). It also fetches solutions from the memories to feed the slaves with new starting solutions. This is more precisely described in the following.

4.1. Communication scheme

The communication scheme is illustrated in Fig. 1 with two slaves, one master, and the two memories associated with SS and VND, respectively. In Phase I, only SS processes are invoked. Initially, each SS process receives a number i between 1 and the number of slaves s in order to set parameter λ to i/s for the initialization of the modified linear costs. The SS processes are used in Phase I to fill the SS and VND memories. Namely, the best solutions produced by the SS processes are stored in the SS Memory (and are used by VND in Phase II). Also, a number of intermediate solutions produced during the iterative SS procedure are stored in the VND memory (and are used by SS in Phase II). More precisely, at each iteration k, the current intermediate solution is stored in memory with probability $1/k$. Intermediate solutions obtained in earlier iterations are thus more likely to be found in the VND memory. The rationale is that SS is more likely to follow a different im-

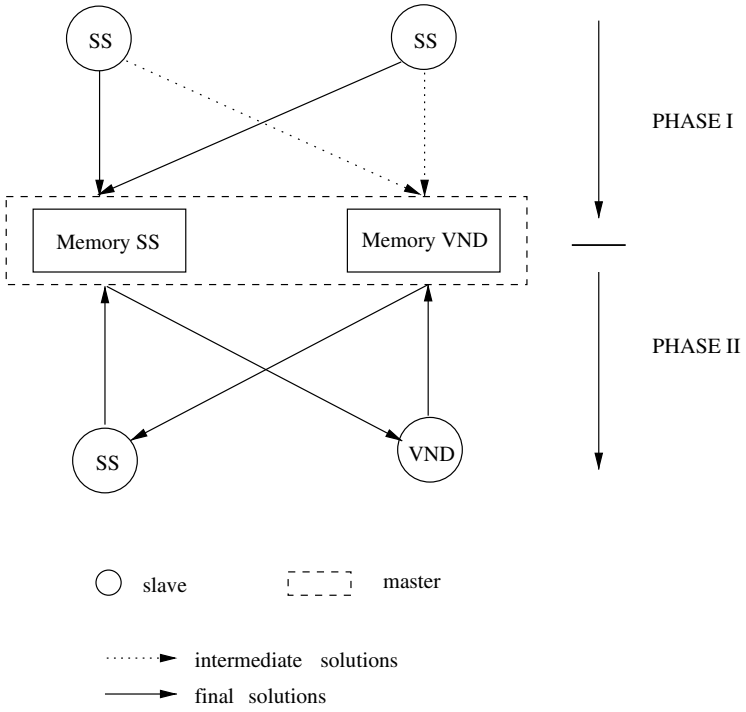

Fig. 1. Parallel search framework.

provement path in Phase II if the intermediate solutions are not too close from the best solution found.

Phase II starts when at least qs solutions have been generated, with q equal to 5 in our computational experiments. At this point, all SS processes terminate and a proportion of them, defined by the user, become VND processes (in the figure, one of the two SS processes becomes a VND process). During Phase II, the VND processes get starting solutions from the SS memory and feed the VND memory with their final solutions. Conversely, the SS processes get starting solutions from the VND memory and feed the SS memory with their final solutions. In the degenerate cases where all processes are SS or VND, the two pools of solutions are virtually merged into a single memory (through a parameter which sets the probability of storing or fetching a solution from one memory or another to 0.5; otherwise, this parameter forces the standard behavior depicted in Fig. 1).

4.2. Adaptive memories

The slaves collaborate by communicating information about new solutions through the SS and VND memories. These memories are adaptive, as they

are continuously updated in order to contain the best solutions found during the search. Namely, a new solution returned by SS or VND is stored in the corresponding memory if:

- the memory is not filled yet;
- the memory is filled, but the new solution is better than the worst solution in memory; in this case, the new solution replaces the worst one.

When a solution is fetched from a memory to provide a new starting point for SS or VND, a probabilistic selection scheme, biased in favor of the best solutions, is used. Assuming l different solutions in memory, the best one (rank 1) is associated with some $Z_{\max}$ value, while the worst one (rank l) is associated with some $Z_{\min}$ value. The values associated with the remaining solutions are then equally spaced between $Z_{\min}$ and $Z_{\max}$. More precisely, the value Z_i for the solution of rank i is computed as:

$$Z_i = Z_{\max} - (Z_{\max} - Z_{\min})\frac{i-1}{l-1}, \quad 1 \leqslant i \leqslant l.$$

The probability p_i of selecting the solution of rank i is then:

$$p_i = \frac{Z_i}{\sum_{j=1}^{l} Z_j}, \quad 1 \leqslant i \leqslant l.$$

Assuming that $Z_{\min} + Z_{\max} = 2$, the selection bias in favor of the best solution can be increased by setting the $Z_{\max}$ value closer to 2, or reduced by setting its value closer to 1 [18]. In the current implementation, each memory contains at most $l = 15$ solutions, with $Z_{\min} = 0.5$ and $Z_{\max} = 1.5$.

To favor diversification, a starting solution taken from a memory is first slightly perturbed before being processed by SS or VND. Namely, a number of randomly chosen depots (among the closed ones) are opened. Similarly, a number of randomly chosen depots (among the open ones) are virtually closed by putting a high penalty on their incident arcs. This is done by multiplying the cost of each arc by 1000. In the latter case, the depots are not definitively closed, to prevent infeasibility due to insufficient capacity. The number of depots to be opened, as well as the number of depots to be virtually closed, is chosen in the interval [5,10] (we assume that all instances have more than 10 depots). After this perturbation, the associated MMCF is solved to determine the initial flow structure and its associated depot configuration.

5. Computational results

In this section, computational results are reported on a set of randomly generated test problems. First, the problem instances are introduced; then, the performance of the parallel implementation is reported. All tests were run on 400 MHz Utra-Sparc II processors, with an individual processor dedicated to each process. The

MPI software (from Sun HPC Cluster Tools 3.1) was used to handle the communications among the processes. The MMCF subproblems were solved with CPLEX 6.6 [11].

5.1. Problem instances

The problem generator for creating the test instances if fully described in [8]. In the following, we summarize the main points.

Customer and depot locations. The rectangular area of interest is divided into a certain number of customer zones, where each zone is a 100×100 square in the Euclidean plane. The total number of customers is distributed among the zones, depending on their type (i.e., high, medium or low-density). Depot zones are centered at the intersection of four customer zones; the depots are then distributed among the depot zones according to the density type of the four intersecting customer zones. Fig. 2 is an example based on a $n_h \times n_v = 3 \times 2$ grid of customer zones (and with the depot zones shown with dashed lines).

Linear and fixed costs. The fixed cost to open and operate a depot is set to a basic value randomly chosen in the interval [1000, 2000]. This value is then multiplied by a scaling factor s_f to put more or less emphasis on the fixed costs versus the linear costs. The linear cost for each arc–commodity pair is based on the Euclidean distance multiplied by a commodity-dependent factor which, for each commodity, is an integer randomly chosen between 1 and 10 (for a given commodity, the same number is used on all arcs). For customer–depot or depot–customer arcs, the linear cost is actually the Euclidean distance between the two nodes multiplied by the commodity-dependent factor. For depot–depot arcs, the linear cost is the Euclidean distance multiplied by the commodity-dependent factor and by 0.6, to take into account the economies of scale obtained through consolidation.

Demands and supplies. Each customer zone is defined as a supply, demand or balanced zone. These three types are distributed among the high, medium and

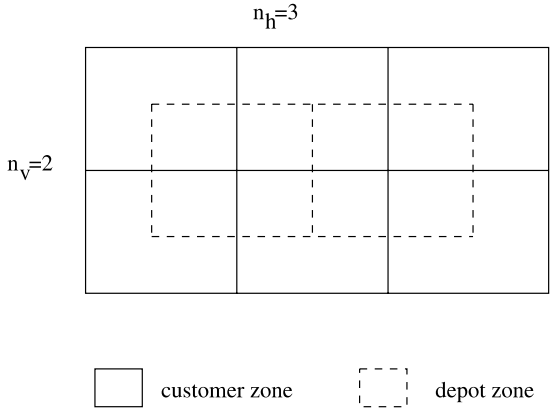

Fig. 2. Customer and depot zones.

low-density zones, at their pro-rata. A mean demand value $\bar{d}$, which is the same for all commodities, is used to assign a demand d_i^k (or supply s_i^k) to each customer i for commodity k. Basically, $\bar{d}$ is randomly perturbed by 10% to 30% to generate different values (see [8] for details).

Capacities. The volume of one unit of commodity is an integer randomly chosen between 1 and 20, for each commodity $k \in K$. Assuming that $V = \sum_{k\in K} v^k \sum_{i\in C^s} s_i^k$ is the maximum volume that might transit through all depots, the capacities are generated in such a way that $\sum_{j\in D} q_j \approx V/\gamma$, where $\gamma \in (0,1)$ is a user-supplied parameter. When $\gamma \to 1$ the problem is more constrained and conversely. Values between 0.2 and 0.4 are indicated: higher values lead to infeasible instances, while lower values lead to virtually uncapacitated problems.

Based on the above characteristics, the following parameter values were used to generate the problem instances:

- number of customers (n): 500,
- number of depots (m): 200,
- number of commodities (p): 20,
- number of customer zones ($n_h \times n_v$): 3×2 (dense), 4×3 (sparse),
- fixed cost multiplier (s_f): 1000, 5000,
- mean demand value for each commodity ($\bar{d}$): 100,
- capacity tightness parameter (γ): 0.3.

Thus, four different types of problems are found in the test set, depending on the number of customer zones and the fixed costs for operating the depots. For each type of problems, there are 10 different instances.

5.2. SWAP neighborhood

One parameter of VND that deserves a particular attention is the number of SWAP moves that are evaluated exactly at each iteration. The algorithm is sensitive to this value, because SWAP is aimed at identifying a good configuration for a given number of open depots. Preliminary experiments have shown that a neighborhood size of about 60 is indicated for the instances under consideration. Smaller values do not allow a sufficiently aggressive search, while larger values are too computationally expensive and severely limit the total number of iterations. In fact, we have chosen a value of $8 \times 8 = 64$. Here, a SWAP is seen as a DROP followed by some ADD move. Consequently, we consider the eight best depots to be closed (based on DROP approximations [8]), and for each one of them, we consider the eight best depots to be opened (based on ADD approximations [8]). Those 64 SWAP moves are then evaluated exactly with CPLEX.

5.3. Distribution of processes

In this subsection, we explore the impact of various distributions of processes among SS and VND during Phase II. To this end, 16 processors have been used

Table 1
Results with different SS/VND process distributions

Problem type	SS/VND									
	1/0		0.75/0.25		0.5/0.5		0.25/0.75		0/1	
	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best
3×2_1000	0.41	0.40	0.31	0.23	0.34	0.28	0.29	0.18	0.30	0.23
3×2_5000	0.45	0.27	0.32	0.18	0.30	0.20	0.25	0.09	0.30	0.19
4×3_1000	0.33	0.32	0.24	0.16	0.26	0.18	0.21	0.16	0.22	0.13
4×3_5000	0.45	0.38	0.28	0.16	0.25	0.18	0.25	0.14	0.28	0.21
Overall	0.41	0.34	0.29	0.18	0.29	0.21	0.25	0.14	0.28	0.19

(plus the master). The following SS/VND distributions have been considered, where each number is the fraction of processes dedicated to SS or VND, respectively: 1/0, 0.75/0.25, 0.5/0.5, 0.25/0.75 and 0/1. Note that the distributions 1/0 and 0/1 correspond to degenerate cases where all processes are SS or VND, respectively. In every experiment, the search was stopped after 1 h of wall clock time. The results are reported in Table 1. The first column indicates the problem type *a_b*, where *a* corresponds to the $n_h \times n_v$ customer zones and *b* to the scaling factor s_f for the fixed cost. The remaining columns represent the various SS/VND process distributions.

Ten instances were generated for each type of problems and three different runs were performed on each instance. The columns *Avg.* show the average of the 30 runs performed over the 10 instances for each problem type, while the columns *Best* refer to the average of the best run over each instance for each problem type. The last line *Overall* is the average taken over all problem types. The values shown correspond to the gap with the optimum (in percent). The latter was obtained with the parallel branch-and-bound algorithm of CPLEX running on 16 processors for total CPU times ranging from 120,784 s (approximately 34 h) to 10,000,000 s. In the latter case, CPLEX was deliberately stopped after that amount of time on two specific instances of type 3×2_5000, without reaching or proving optimality.

The results in Table 1 indicate that (1) VND is essential for good solutions to emerge and (2) combining SS and VND processes is beneficial, as the distribution 0.25/0.75 consistently leads to better results (except in the case of column "Best" for problem type 4×3_1000). Additional experiments with 4 and 8 processors were also performed with similar results. The distribution 0.25/0.75 was thus selected for the next set of experiments.

5.4. Number of processors

The aim of this section is to study the impact of the number of processors on solution quality. That is, for the same computational effort, is it better to work with

Table 2
Results with different numbers of processors

Problem type	16 processors		8 processors				4 processors			
	1 h		1 h		2 h		1 h		4 h	
	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best
3×2_1000	0.29	0.18	0.35	0.30	0.32	0.26	0.42	0.36	0.36	0.31
3×2_5000	0.25	0.09	0.33	0.17	0.27	0.14	0.31	0.21	0.23	0.09
4×3_1000	0.21	0.16	0.27	0.22	0.26	0.21	0.30	0.22	0.27	0.19
4×3_5000	0.25	0.14	0.32	0.22	0.31	0.18	0.38	0.29	0.34	0.25
Overall	**0.25**	**0.14**	0.32	0.23	**0.29**	**0.20**	0.35	0.27	**0.30**	**0.21**

a smaller or a larger number of processors? In the following, a new set of experiments was performed with 4, 8 and 16 processors (plus the master). To obtain an equivalent amount of computational work, the experiments with 16 processors were run for one hour of wall clock time, while those with 8 and 4 processors were run for two and four hours of wall clock time, respectively, always using the 0.25/0.75 distribution. The results are summarized in Table 2, using a format similar to Table 1.

As indicated in bold, an improvement is observed with an increasing number of processors. Note that more processors running in parallel allows for a higher "refreshment" rate of the adaptive memories, thus leading to a more diversified search. Given that SS is combined with a relatively simple VND local search, it is particularly important for the adaptive memories to provide starting points that represent a good sampling of the search space. This is better achieved with a larger number of processors.

Finally, it is worth noting that the proposed approach has allowed us to find better solutions than those obtained in [8] with a tabu search heuristic. In the latter case, the best reported gaps on problem types 3×2_1000, 3×2_5000, 4×3_1000 and 4×3_5000 were equal to 0.31%, 0.18%, 0.33% and 0.29%, respectively (often using substantially larger computation times).

6. Conclusion

A parallel hybrid framework, involving SS and VND, was developed to address a multicommodity capacitated location problem with balancing requirements. The proposed framework introduces a new cooperative scheme where both SS and VND procedures feed each other with new starting solutions through adaptive memories. The results show that combining SS and VND within the parallel hybrid framework provides better quality solutions than SS or VND alone. Also, for the same amount of computational work, a larger number of processors is indicated,

as it leads to a more diversified search. The proposed approach has allowed us to find the best solutions to date on our test instances.

Acknowledgements

Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is gratefully acknowledged. Thanks also to Serge Bisaillon for running the computational experiments.

References

[1] P. Badeau, F. Guertin, M. Gendreau, J.-Y. Potvin, E.D. Taillard, A parallel tabu search heuristic for the vehicle routing problem with time windows, Transportation Research C 5 (1997) 109–122.
[2] T.G. Crainic, P.J. Dejax, L. Delorme, Models for multimode multicommodity location problems with interdepot balancing requirements, Annals of Operations Research 18 (1989) 279–302.
[3] T.G. Crainic, L. Delorme, Dual-ascent procedures for multicommodity location–allocation problems with balancing requirements, Transportation Science 27 (1993) 90–101.
[4] T.G. Crainic, L. Delorme, P.J. Dejax, A branch-and-bound method for multicommodity location with balancing requirements, European Journal of Operational Research 65 (1993) 368–382.
[5] T.G. Crainic, M. Gendreau, P. Soriano, A tabu search procedure for multicommodity location/allocation with balancing requirements, Annals of Operations Research 41 (1993) 359–383.
[6] B. Gendron, T.G. Crainic, A branch-and-bound algorithm for depot location and container fleet management, Location Science 3 (1995) 39–53.
[7] B. Gendron, T.G. Crainic, A parallel branch-and-bound algorithm for multicommodity location with balancing requirements, Computers & Operations Research 24 (1997) 829–847.
[8] B. Gendron, J.-Y. Potvin, P. Soriano, A Tabu Search with Slope Scaling for the Multicommodity Capacitated Location Problem with Balancing Requirements, Technical Report CRT-2001-37, Centre de Recherche sur les Transports, Montréal, 2001.
[9] B. Gendron, J.-Y. Potvin, P. Soriano, Tabu search with exact neighbor evaluation for multicommodity location with balancing requirements, INFOR 37 (1999) 255–270.
[10] P. Hansen, N. Mladenović, An introduction to variable neighborhood search, in: S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Boston, 1999, pp. 433–458.
[11] ILOG CPLEX 6.6, 1999.
[12] D. Kim, P.M. Pardalos, A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure, Operations Research Letters 24 (1999) 195–203.
[13] D. Kim, P.M. Pardalos, Dynamic slope scaling and trust interval techniques for solving concave piecewise linear network flow problem, Networks 35 (2000) 216–222.
[14] D. Kim, P.M. Pardalos, A dynamic domain contraction algorithm for nonconvex piecewise linear network flow problems, Journal of Global Optimization 17 (2000) 225–234.
[15] Y. Rochat, E.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147–167.
[16] R. Sridharan, The capacitated plant location problem, European Journal of Operational Research 87 (1995) 203–213.

[17] E.D. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, A tabu search heuristic for the vehicle routing problem with soft time windows, Transportation Science 31 (1997) 170–186.
[18] D. Whitley, The GENITOR algorithm: why rank-based allocation of reproductive trials is best, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, 1989, pp. 116–121.